# MEAD User Manual - Version 2.0

## *Software Usage Notice*

By downloading this software the user agrees to accept this software entirely as is, with absolutely no warranty whatsoever, expressed or implied. The responsibility for ensuring fitness and correctness for any purpose lies entirely with the user.

Questions about this document to be sent to mead-support@lists.andrew.cmu.edu

## *Index*



## *What is MEAD?*

The Middleware for Embedded Adaptive Dependability (MEAD) infrastructure aims to enhance distributed real-time middleware applications with new capabilities including:

1. Transparent, yet tunable, fault-tolerance in real time
2. Proactive dependability
3. Resource-aware system adaptation to crash, communication and timing faults
4. Scalable and fast fault-detection and fault-recovery

The MEAD 2.0 release currently includes the replication framework for Active replication. Users interested in the MEAD Manager should refer to the MEAD 1.5 release.

More information, including publications on the MEAD system, can be found at the MEAD website (http://www.ece.cmu.edu/~mead).

## Fault-Tolerance Fundamentals

This release of MEAD focuses on non-adaptive replication, i.e., the replication style does not currently change on the fly. Given the current state-of-the-art in ORBs, which are not completely stateless black-boxes, the unit of replication (also known as a replica) is effectively the process or the container, and not the individual object or the component. Note that replicating processes automatically replicates the components/objects hosted within those processes. The replication styles currently offered by MEAD are active replication and warm passive replication.

From Figure 1, Active replication allows multiple replicas when spawned, to join a replica group. Each individual replica processes each invocation and sends results back to the client. The beauty lies in the fact that MEAD suppresses duplicate responses, and the client believes it is communicating with a single server. Checkpoints (i.e., state snapshots) are triggered in active replication only when a new replica is launched and needs to synchronize its state with those of the other running replicas.

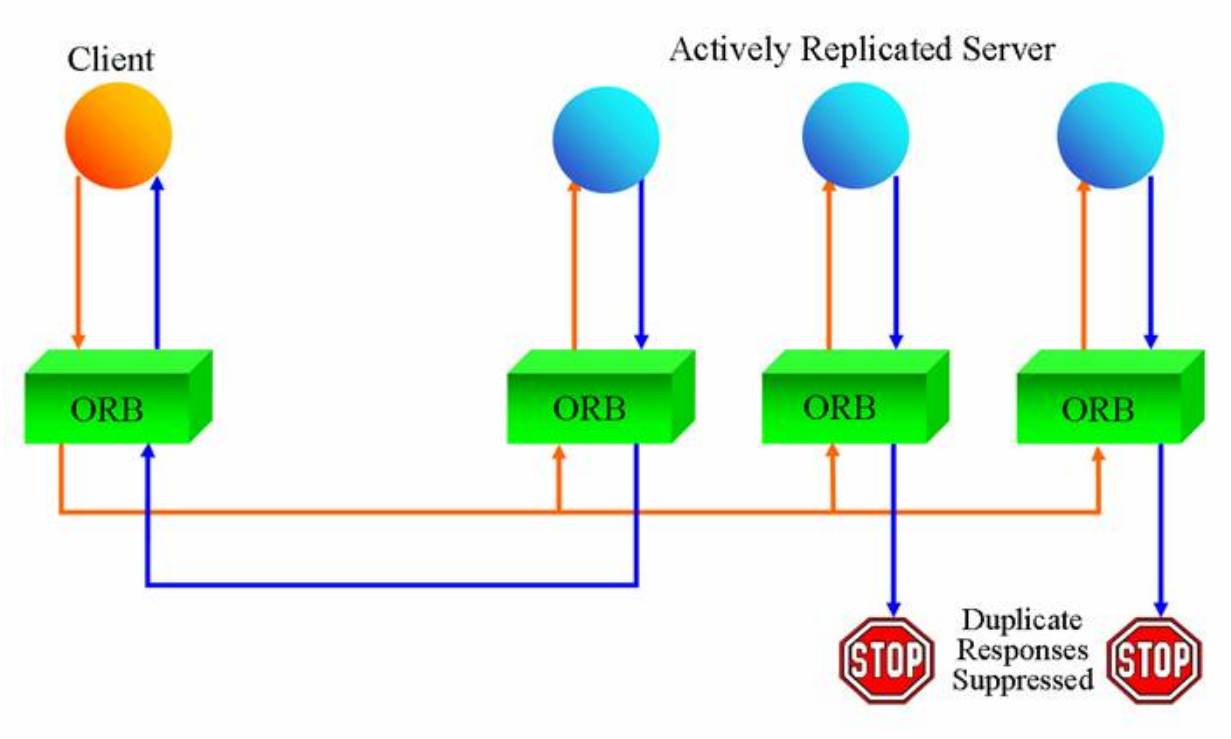


***Figure 1 – Active Replication***

From Figure 2, Warm passive replication designates one replica to be the primary and the others to be backups, in the event of the primary's failure. Only the primary replica processes invocations from the client and responds to the client. All backups in the replica group periodically receive checkpoints from the primary that allow them to synchronize their respective states with that of the primary.

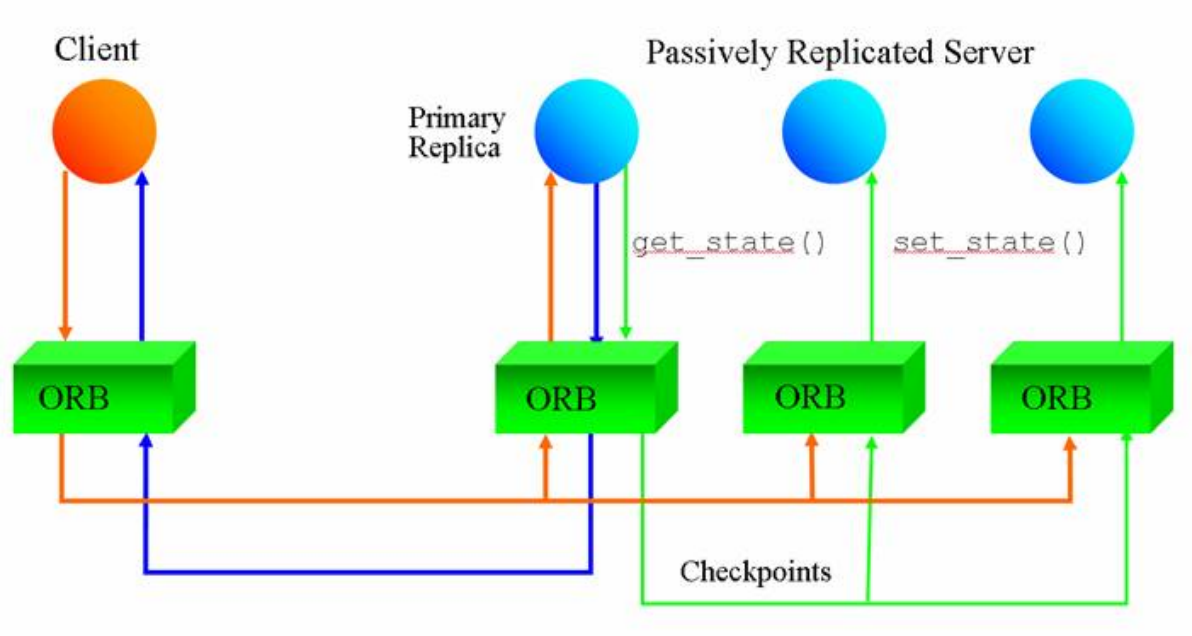


***Figure 2 – Warm Passive Replication***

Both replication styles offer fault tolerance through redundancy. Active replication can be resource intensive (because of increased CPU usage and bandwidth usage), but provides faster response and recovery times. On the other hand, warm passive replication typically conserves resources, but the response time is bounded by the network and processing speed of the primary replica. Also, increased bandwidth and CPU usage might result if the size and frequency of checkpoints is sufficiently large. Warm passive replication's recovery times are also slower because the recovery time includes failover time from the crashed primary to the backup.

## *Installation and Distribution*

### Release Notes

The following list outlines functionality and enhancements of the current and past MEAD releases as a change log:

MEAD v2.0

- Complete rewrite of the replication framework
- Support for mixed TCP and GCS clients, so called Singleton concept
- At this time the MEAD Manager has not been integrated
    - The MEAD Manager can be found in the MEAD 1.5 release.

MEAD v1.5

- Resource monitoring integrated
- Automatic replica launch and bootstrapping
- 100% improvement in Active Replication client side round-trip latencies
- Several minor bug fixes and enhancements

MEAD v1.1

- Added support for the Naming Service
- Added support for Spread Messages up to 80KB
- Improved RTT latency by ~600usec
- Several minor bug fixes and enhancements

MEAD v1.0

- Initial MEAD release, replication framework only
- Support for Active or Warm Passive Server Replication
- Support for CORBA IOR files

We recommend using MEAD 2.0 (which is a stable release)

## Currently Supported Configurations

Emulab (http://www.emulab.net) is the distributed test-bed of choice for the MEAD system. If you have never used Emulab before, please refer to the Emulab "Getting Started" Tutorial (http://www.emulab.net/tutorial/docwrapper.php3?docname=tutorial.html). More details about MEAD running on Emulab can be found below in the Running MEAD on Emulab section.

The current version of MEAD supports, and has been tested with, the following platforms, ORBs and third-party software.

| | |
|---|---|
| Platform | Fedora Core 4 (ARMS FC4-UPDATE image on Emulab) |
| Compiler | g++ 4.0.2 |
| Network | 100Mbps Ethernet |
| CORBA Version | TAO v 1.5.3 and ACE v 5.5.3 |
| Spread Version | Spread group communication system v 4.00.00 |

***Table 1 – Supported Configuration***

MEAD uses the Spread group communication system for inter-object communication. More details can be found below under the Running and Configuring Spread section of this manual.

## Distribution

The directory containing the latest release of the MEAD software distribution is located on the users server at Emulab (users.emulab.net) at /groups/pces/uav_oep/mead_cmu/release/mead.

The current MEAD release (2.0) is comprised of library code for the MEAD replication mechanism, pre-built Spread libraries for FC4, and example scripts used to run the replication infrastructure. Figure 3 displays the directory structure.

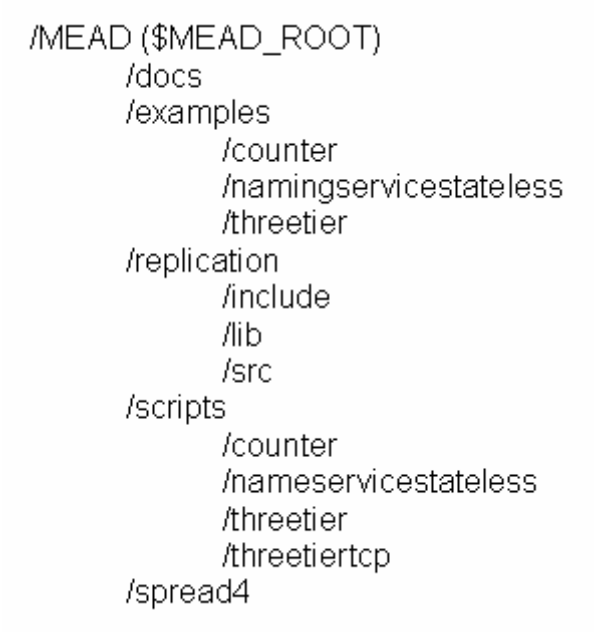


***Figure 3 – MEAD Directory Structure***

## Known Caveats and Recommendations

In the interests of fault containment and effective replication, we recommend hosting only one replica of a server per node. While it is possible to host more than one server replica per node a single processor-crash may affect more than one replica.

The current version of MEAD requires the CORBA application to be deterministic in behavior, i.e., any two replicas of the server, when starting from the same initial state, and receiving the same set of invocations in the same order, should reach the same final state. This eliminates the use of local timers, shared memory, threading, and any other OS primitives that can lead to irreproducible behavior across different nodes in a distributed system. Determinism is a common assumption in the development of fault-tolerant distributed systems; while ongoing development in the MEAD project aims to eliminate the need for determinism, the current version of MEAD requires this of the application.

## Reporting Problems & Obtaining Support

If you face a specific problem with running or installing the MEAD system please email us at mead-support@lists.andrew.cmu.edu

# *CORBA Application Requirements*

In order to restore application-level state in the event of a server crash, the Fault-Tolerant CORBA standard requires every CORBA object to support an additional Checkpointable interface, with methods for the retrieval and assignment of application-level state. This interface

is an abstract class and can be inherited by the application when working with MEAD. Note that State can be defined in a custom way, based on the CORBA object's state. MEAD simply invokes these methods in order to perform checkpointing and state transfer for both warm passive and active replication. After implementing these functions the application programmer does not need to worry further about them (except, of course, for updating the implementations of these methods should the application's definition of state change) MEAD handles ORB-level and infrastructure-level state so that the application programmer does not need to worry about them.

```
exception NoStateAvailable{};
exception InvalidState{};

interface Checkpointable {
  State get_state() raises(NoStateAvailable);
  void set_state(in State s) raises(InvalidState);
};
```

***Figure 4 – Object Checkpoints***

When developing CORBA applications for MEAD, it is necessary to implement *get_state()* and *set_state()* at the object level, and *get_global_state()* and *set_global_state()* for void pointer data at the application level (extern functions).

At the application level, the extern'd functions in Figure 5 are called by MEAD. The void data can be any type of structure containing all of the current state for the process as well as any individual CORBA object state show in Figure 4. Figure 5 shows C prototypes that can be used to declare the external functions *get_global_state()* and *set_global_state()*:

```
extern "C" void *get_global_state (int *size);
extern "C" void set_global_state (void * state, int size);
```

***Figure 5 – Extern State Functions***

The sample program "counter" ($MEAD_ROOT/examples/counter) contains simple usage of both the Checkpointable interface and the globally extern'd functions.

## *Building MEAD*

The MEAD replication library requires Spread GCS for communication, more details can be found further in this document under Running and Configuring Spread. Spread must be installed and correctly configured prior to building MEAD. More details can be found at the Spread website (http://www.spread.org). Example programs require ACE/TAO to be installed, details can be found at the TAO website (http://www.cs.wustl.edu/~schmidt/TAO.html).

Before building the MEAD release and sample programs source the 'source.sh' file (% source source.sh). This file will establish the $MEAD_ROOT and $SPREAD_HOME variables that are needed to run sample programs and builds. The source.sh file may require slight tweaking for your environment. After the source completes, a recursive make can be ran in the $MEAD_ROOT.

The replication makefile makes use of the Spread library (libspread.so) found in the spread4 directory. If you are not using FC4 it will be necessary to install and configure Spread on your target platform. For convenience this MEAD distribution includes pre-built Spread binaries for FC4 in the spread4 directory.

Finally, both the sample CORBA applications as well as the MEAD Replication Library makefiles can compile the target application in debug mode. This will enable debugging messages, displaying the inner workings of the MEAD Replication Library or CORBA applications on the standard output.

### To Enable Debugging

To enable debugging of the MEAD replication library ($MEAD_ROOT/replication/makefile) look for the CPPFLAGS variables and values, they will be commented out by default. Two values can be used for runtime debugging: −DDEBUG and −DTRACE. −DDEBUG is used for various print statements containing runtime values, while −DTRACE is used for a trace some of the function calls in the replication library. Uncomment the values and recompile to view debugging information.

Makefiles for the example programs also contain debugging information. Look for and uncomment the mINCLUDES variable that contains the −DDEBUG value.

## *Running and Configuring Spread*

### Spread Overview

Spread is a group communication system developed at John Hopkins University and currently developed by Spread Concepts LLC and CNDS at John Hopkins University. Additional information can be found at the Spread website (http://www.spread.org) and the Spread User Manual.

The Spread daemon can be launched using this command:

```
% spread -c <spread.config.file>
```

### Spread Segment

The Spread configuration file contains options for the Spread daemon, and comments on the configuration of each option. MEAD is mostly concerned with the Spread_Segment { }, which contains a list of <hostname> <IP> pairs for each node in the cluster, as well as a broadcast address and port for execution. The following is a sample configuration file used in the Emulab environment. The broadcast address can differ in range (i.e., in the first three dotted-decimal places) from the IP addresses of the three nodes; this is specific to the Emulab environment where the nodes seem to be multi-homed. The broadcast is for local connectivity and the individual host IPs are actual addresses that can be resolved for the machine.

```
Spread_Segment BroadcastAddress:SpreadPortNumber {
        node0   node0_IP_address
        node1   node1_IP_address
        node2   node2_IP_address
}
```

***Figure 6 – Spread Segment***

Finally, only use the computer name for the <Hostname> parameter and not the fully qualified name for the computer (e.g. use node0 not node0.test.pces.emulab.net).

### Spread Timeouts

When compiling the Spread library and the Spread daemon, there is a series of timeouts that can be set can be configured and compiled. These can significantly affect the performance of both Spread and MEAD. More details can be found in the Spread User Manual.

### Spread Related Errors returned by MEAD

The following lists a few common errors related to configuration of spread that are returned from the MEAD library, more details can be found at: http://www.spread.org/docs/docspread.html. When launching spread from the Spreadstart script in the $MEAD_ROOT/scripts folder errors are generally masked by the scripts. To trouble shoot the issue, launch spread at the command line as shown above.

- MEAD: Could not connect to SPREAD daemon! (error=-2)
    - The Spread_Segment may have not been setup properly to include the <Hostname> <IP> where the current application resides.
    - The port used in the Spread_Segment may be different from the port used in the MEAD library

## *Running MEAD on Emulab*

The steps below detail the execution of the MEAD replication library using example bash scripts provided in the $MEAD_ROOT/scripts directory. This run will replicate three servers in active replication styles and will use a single client to make requests of the three replicated server. The

following bash scripts set environmental variables necessary to execute MEAD, open them up and take a look at how the variables are used.

The current test-bed of choice for the MEAD software is Emulab (FC4-UPDATE Emulab image).

Some of the following steps need to be executed on every assigned Emulab node (step is prefaced by [every node]) or only on one assigned Emulab node (step is prefaced by [one node]). The symbol % represents the shell command-line prompt.

1. Setting up the Emulab Experiment
    a. Create a new Emulab experiment consisting of four nodes, one 100Mbps LAN; three nodes will be used to host three individual server replicas, and the fourth node will be used for the client application.
    b. If possible use the ARMS OSID FC4-UPDATE for each of the nodes when setting up your experiment with Emulab, otherwise you'll need a comparable image running Fedora Core 4 and containing the RPMs for the current ACE/TAO release shown in Table 1 above.
    c. Wait to receive an email from the Emulab Testbed Operations team (notifying you of the availability of your requested experimental configuration) before proceeding further.
    d. Once you receive the notification about your experiment being started, ssh into the machines assigned to you, according to the instructions on Emulab.
2. **[any node]** Obtaining the MEAD source distribution
    a. The MEAD distribution is available on users.emulab.net in the directory: /groups/pces/uav_oep/mead_cmu/release/mead/
    b. Use cp (or scp) to copy the MEAD source to your working directory (for convenience, copy from the tar ball from the root of the release folder).
    c. If you copied the .tar.gz file, then, you should do the following to extract the MEAD directory: % tar -zxf mead-1.0.tar.gz
    d. A directory called MEAD is created in the process of extracting the .tar.gz file.
3. **[every node]** Obtaining the right environmental settings
    a. Enter the MEAD directory and source the source.sh file to establish some environmental settings for your binary and library search-paths:
    % source source_this_for_Emulab
4. **[any node]** Building the MEAD library and the test applications
    a. Enter the MEAD root and type make to run a recursive make:
    % cd $MEAD_ROOT
    % make
5. Setting up the Spread configuration
    a. For each of your assigned Emulab nodes, query the IP address, the hostname, and the broadcast address.
        i. **[any node]** Emulab machines are multi-homed (five interfaces per node). To see this for yourself, try the following:
            1. % ifconfig -a
            2. You will see interfaces eth0 through eth4 defined.

ii. **[any node]** For the broadcast address, look at the output of:
    1. % ifconfig eth3
    2. and replace the last part of the dotted-decimal IP address with 255, e.g., if you see an IP address of 10.1.1.4, the broadcast address is 10.1.1.255
iii. **[every node]** For each node's IP address, look at the output of:
    1. % hostname -i
    2. to obtain the IP address associated with the interface.
iv. **[every node]** For each node's hostname, look at the output of:
    1. % hostname
    2. to obtain the qualified hostname associated with the interface. You will only need the first part of this address, i.e., for node8. *.arms.emulab.net, you only need to remember node8).

b. **[any node]** Edit the file MEAD/scripts/spread.conf.emulab to specify the set of Emulab nodes that you are using for your experiment.
    i. Edit the Spread_Segment by listing the <hostname> <ip> pairs within the parentheses.
    ii. The default Spread port number is 6011, but can be changed; however, the new value of the port must be identical in two places: (i) the Emulab configuration file MEAD/scripts/spread.conf.emulab, and (ii) the SPREAD_PORT environment variable supplied to the application (and, indirectly, to MEAD) at run-time.
    iii. Here is an example for a broadcast address of 10.1.1.255, a Spread port number of 6011 and four nodes (node6, node7, node8 and node9) with their respective IP addresses:

```
Spread_Segment 10.1.1.255:6011 {
  node6  155.101.132.92
  node7  155.101.132.93
  node8  155.101.132.94
  node9  155.101.132.95
}
```

6. **[every node]** Running Spread
    a. Launch the Spread daemon (spread) on each host that will run either a server or a client. The Spread daemon can be launched from the MEAD root directory, as follows: % cd MEAD/scripts
    b. % ./Spreadstart
7. There are several scripts that demonstrate the execution of MEAD. All of the scripts are found in the MEAD/scripts directory and display usage information. The scripts will not launch if the Spread daemon is not running. The scripts provide settings for most of the MEAD environment variables. If you wish to bypass the scripts and run processes directly at the command-line, please read the MEAD Replication Library Parameters section of this document.

## Must-Read Notes

Some common things to watch out for when running MEAD:

- The source.sh file must be sourced only from the $MEAD_ROOT directory, where it is located.
- The Spread daemon must be running on every node on which you expect to run clients or servers using MEAD.
- The same Spread group id (GID) should only be used once per host.
- When replicating a server, ensure that all of the replicas of the server are launched with the same object group identifier and the same replication style, i.e., within the same group, all of the replicas must possess the same GID and should have the same REPLICATION_STYLE
- Pure clients should always be launched using ACTIVE replication style in order to enable duplicate suppression (the client scripts in the $MEAD_ROOT/scripts directory automatically take care of this), regardless of the server's replication style.
- The error message including “from LD_PRELOAD cannot be preloaded: ignored.” implies the MEAD replication library hasn’t been build or can’t be found in the path specified. See scripts for examples.
- The singleton concept allows the bridging of TCP and GCS clients into GCS domains. In this case there is only a single “singleton” node, multiple mixed clients, and one or more replicated servers. The singleton functions as a bridge.

## *Using the Naming Service*

As of MEAD 2.0, support for the Naming Service is handled with the use of the BYPASS list. In order to connect to the Naming Service the BYPASS list must include an ip:port listing for the Naming Service.

For example, when launching a simple application, communicating with the Naming Service at IP 192.168.1.1 and port 6767 try the following:

```
% env LD_PRELOAD=$MEAD_ROOT/replication/libmead.so.1 \
   GID=5555 \
   BYPASS=192.168.1.1:6767, \
   $MEAD_ROOT/examples/counter/client iorfile time
```

## *MEAD Replication Library Parameters*

With the rewrite of MEAD 2.0 there has been drastic changes to the library parameters, Table 2 displays the updated environmental variables for the MEAD library.

| Name | Status | Description | Value |
|---|---|---|---|
| LD_PRELOAD | Required | library path to the MEAD replication library | libmead.so.1 |

| | | | |
|---|---|---|---|
| GID | Required | Integer. Valid dynamic port | 1024 – 65535 |
| REPLICATION_STYLE | Optional | ACTIVE is default. Determines replication style. | ACTIVE or WARM |
| REPORT_CALLS | Deprecated | | |
| SERVER_ID | Deprecated | SERVER_IDs are now determined dynamically through use of the IOR | |
| SERVER_ID2 | Deprecated | SERVER_IDs are now determined dynamically through use of the IOR | |
| IS_STATELESS | Optional | NO is the default. When set to YES MEAD will not call checkpointing functions. | YES or NO |
| SPREAD_PORT | Optional | Integer. Valid dynamic port. By default this is set to 6011 | 1024-65535 |
| USING_NS | Deprecated | Naming Service communication is now achieved by using the BYPASS variable. | |
| NS_HOST | Deprecated | Naming Service communication is now achieved by using the BYPASS variable. | |
| NS_PORT | Deprecated | Naming Service communication is now achieved by using the BYPASS variable. | |
| MANAGER_PORT | Not Avail | MEAD Manager is not currently incorporated in the 2.0 release. | |
| HOSTNAME | Not Avail | MEAD Manager is not currently incorporated in the 2.0 release. | |
| BYPASS | Optional | BYPASS is a list containing one or more ip:port combinations that will bypass GCS communication, thus preforming out-of-band. | IP:PORT,IP:PORT,… |
| IS_SERVER | Deprecated | This is now determined on the fly. | |
| IS_SINGLETON | Optional | Indicates this replica is a gateway between replicated and mixed TCP/GCS domains. A common use is to employ a singleton when bridging only TCP clients | YES or NO |

| | | | |
|---|---|---|---|
| | | (and possibly GCS clients) to redundant GCS tier. Singletons can not be launched replicated. | |
| USES_CALLBACK | Not Avail | CALLBACK consistency is not currently implemented in the 2.0 release. | |
| IS_BLOCKING | Not Avail | Blocking | |
| ENABLE_BATCH | Not Avail | | |

***Table 2 – MEAD library parameters***

## *Other Configurations*

MEAD will run in other environments that have configured functional copies of both ACE/TAO and Spread. The examples above should work as well provided that:

- The proper paths are set for the environment to build and runtime libraries.
- MEAD makefiles for both the MEAD Replication Library and the sample applications are modified to include the proper paths to libraries and include files
- The MEAD Replication Library is rebuilt using the version of Spread for the intended platform
- The sample CORBA applications are rebuilt using the version of ACE/TAO for the intended platform
- All of the scripts in the scripts sub-directory should be changed to include the proper path for Spread

## *Trouble-shooting*

| Title | Description/Possible Solution |
|---|---|
| LD_PRELOAD is not set | In most cases the CORBA application will still function, without the replication mechanism. |
| When Spread is not running | MEAD will return the following error message: Could not connect to SPREAD daemon! (error=-2). The application will not actually launch. |
| env variables are not correct | Critical environmental variables are checked during initialization. MEAD will return error messages that variables are not set. |
| LD_LIBRARY_PATH does not include right libraries | These are standard errors that will be returned by the run time linker and loader. In most cases they include the name of the library that has not been pathed. |

***Table 3 – Common Problems Running MEAD***

## *References*

Publications on the MEAD sytem: http://www.ece.cmu.edu/~mead
The Spread group communication system: http://www.spread.org/
Emulab distributed test-bed: http://www.emulab.net/
Emulab Tutorial: http://www.emulab.net/tutorial/docwrapper.php3?docname=tutorial.html
Fault-Tolerance CORBA specification: http://www.omg.org/docs/formal/04-03-21.pdf
TAO CORBA: http://www.cs.wustl.edu/~schmidt/TAO.html

## *Contributors*

Contributors to MEAD include:

- Tudor Dumitras
- Priya Narasimhan
- Aaron Paulos
- Soila Pertet
- Charlie Reverte
- Joe Slember
- Deepti Srivastava